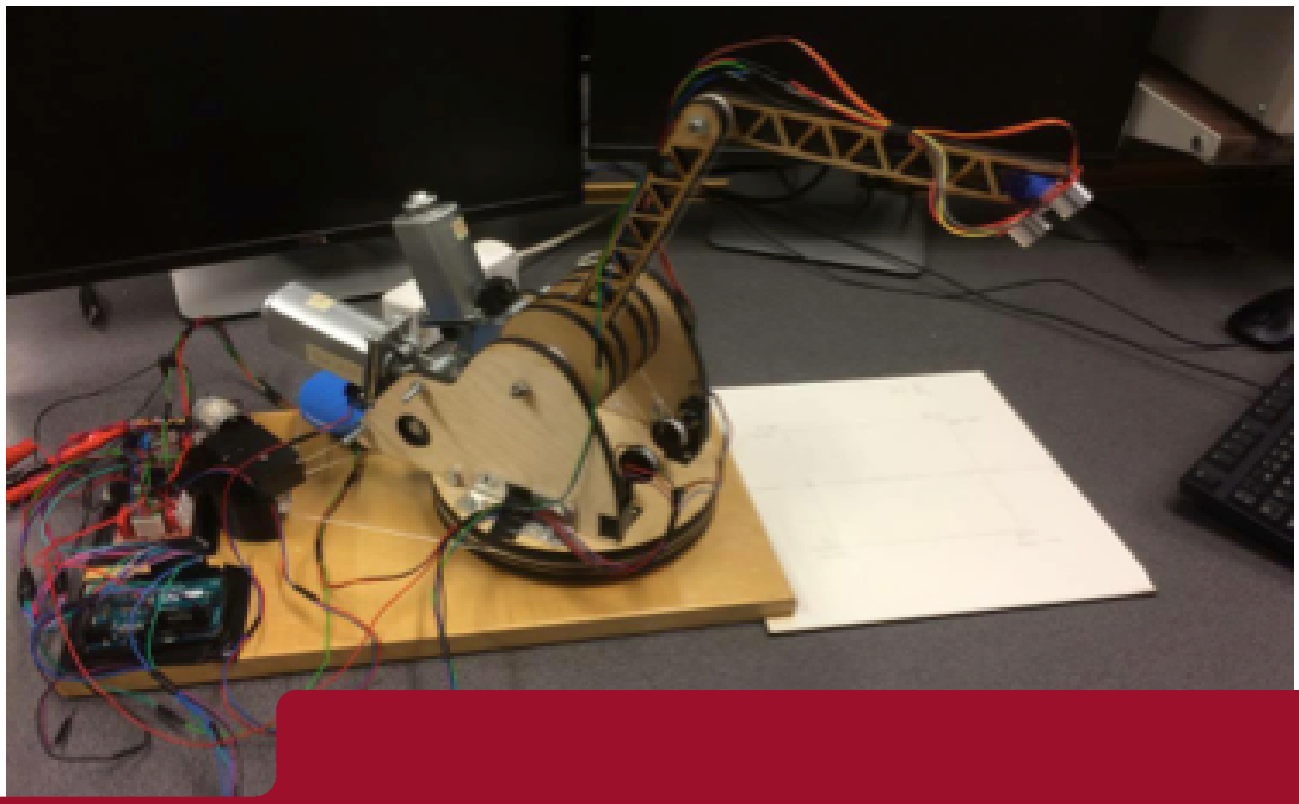
DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
*STOCKHOLM, SWEDEN 2017*

# Feeding Robot


**FILIP MÄLBERG**

**LINDA TRUONG**


**KTH ROYAL INSTITUTE OF TECHNOLOGY**
**SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT**

# Feeding Robot

FILIP MÄLBERG
LINDA TRUONG

Bachelor's Thesis in Mechatronics
Supervisor: Fariba Rahimi
Examiner: Nihad Subasic

TRITA MMK 2017:26 MDAB 644

# Abstract

This project considers a construction of a robot arm that follows a specific pattern to detect food. To thereafter reach a position which illustrates the position of a person's mouth. Due to limited time during this project the robot has no gripper to grab the food. To find food on a specific area, a distance sensor was required. Therefor, an ultrasonic sensor was used where it was fasten to a servo motor. The servo motor enabled the ultrasonic sensor to always face down while searching for food. To achieve a precise motion and position during the movement pattern a PID-regulator was implemented. An experiment was done to compare how different controllers affected the system on different angular distances. With modeling the system and implementing pole placement control the theoretical parameters were calculated. To steer the DC motors a feedback of the current position needed to be known, this was solved using potentiometers. All torque transmission was done by wires, which was an advantage for this project due to the large weight of motors. The final outcome accomplished to follow the given pattern in a suitable way and detect a piece of food.

# Referat

## Matande robot

Denna rapport omfattar en konstruktion av en robotarm och dess rörelse att följa ett specifikt mönster för att detektera mat. Därefter sträcka sig till en position som illustrerar en människas mun. Då avsedd tid för detta arbete var begränsat, konstruerades roboten utan en gripare. För att hitta mat i ett avgränsat område, krävdes en avståndssensor. Därför valdes en ultraljudssensor som fästes på en servomotor. Servomotorn gjorde det möjligt för ultraljudssensorn att alltid vara riktad nedåt under sökandet efter mat. För att uppnå en precis rörelse och position under mönstret, implementerades en PID-regulator. Ett experiment utfördes för jämförelse av hur olika regulatorer påverkade systemet på olika vinkelavstånd. De teoretiska parametrarna beräknades med hjälp av modellering av systemet och polplacering. För att styra DC motorerna måste robotarmens position vara känd, detta löstes med potentiometrar. All kraftmoment överfördes genom vajrar, vilket var en fördel för detta projekt på grund av den tunga vikten på motorerna. Slutresultatet av konstruktionen lyckades följa det givna mönstret och detektera mat på ett lämpligt vis.

# Acknowledgements

We would like to thank our supervisor Fariba Rahimi for support during the project, Tomas Östberg for providing us the important parts for holding up the construction and Staffan Qvarnström for providing the electronic parts and his advices. We are also thankful for the support from the teaching assistants, our FiM classmates and previous FiM classmates that helped us.

# Contents

**Nomenclature**

**Abbreviations**

**1 Introduction** **1**
1.1 Background . . . 1
1.2 Purpose . . . 1
1.3 Scope . . . 2
1.4 Method . . . 2

**2 Theory** **5**
2.1 Past Research . . . 5
2.2 PID-control . . . 7
2.3 Model of the system . . . 8
2.4 PWM (Pulse Width Modulation) . . . 10
2.5 Ultrasonic sensor . . . 10
2.6 Coordinate System . . . 10

**3 Demonstrator** **13**
3.1 Problem formulation . . . 13
3.2 Hardware . . . 14
3.2.1 Chosen parts . . . 14
3.2.2 Design . . . 15
3.3 Electronics . . . 18
3.4 Software . . . 19
3.5 Results . . . 21

**4 Discussion and conclusion** **27**
4.1 Discussion . . . 27
4.2 Conclusions . . . 29

**5 Recommendations and future work** **31**
5.1 Recommendations . . . 31
5.2 Future work . . . 31

**Bibliography** **33**

**Appendices** **34**

**A Datasheets** **35**

**B Arduino Code** **43**

**C Matlab Code** **59**

# Nomenclature

| Symbols | Description |
|---|---|
| $G_{Cl}$ | Closed loop function. |
| $G_C$ | Transfer function of the PID-controller. |
| $G_P$ | Transfer function of the plant. |
| $I$ | Current in Laplace form. |
| $J_{rotor}$ | Moment of inertia of the rotor. |
| $J_{system}$ | Moment of inertia of the system. |
| $K$ | Replacing torque constant and back emf constant. |
| $K_e$ | Back emf constant. |
| $K_t$ | Torque constant. |
| $L$ | Inductance. |
| $L_1$ | Distance of the lower arm. |
| $L_2$ | Distance of the upper arm. |
| $M_{acc}$ | Torque of the acceleration. |
| $M_{load}$ | Torque of the load. |
| $M_m$ | Torque of the motor. |
| $P_{xy}$ | XY-coordinate. |
| $P_x$ | X-coordinate. |
| $P_y$ | Y-coordinate. |
| $P_z$ | Z-coordinate. |
| $R$ | Resistance. |
| $V$ | Voltage. |
| $\ddot{\theta}$ | Angular acceleration. |
| $\dot{\theta}$ | Angular velocity. |
| $\phi_1$ | Angle of the lower arm. |
| $\phi_2$ | Angle of the upper arm. |
| $\tau$ | Variable of integration. |
| $\theta$ | Angle of the motor axle. |
| $\theta_1$ | Angle of the base. |
| $b$ | Viscous friction. |
| $c$ | Speed of sound. |
| $e_{emf}$ | Back emf. |

| Symbols | Description |
|---|---|
| $h_z$ | Height of the lower arm's rotational axle. |
| $i$ | Current. |
| $n_m$ | Gear ratio. |
| $n_{system}$ | Gear ratio of the system. |
| $r$ | Reference value. |
| $s$ | Variable of Laplace transform. |
| $t$ | Time. |
| $u$ | Control signal. |
| $y$ | Output value. |
| $e$ | Error value. |

# Abbreviations

| Acronyms | Description |
|---|---|
| 3D | Three Dimensional. |
| CAD | Computer-aided design. |
| DC | Direct current. |
| GND | Ground. |
| MATLAB | Matrix Laboratory. |
| PID | Proportional–Integral–Derivative controller. |
| PLA | Polylactic acid. |
| PWM | Pulse Width Modulation. |

# List of Figures


2.1 Rotational joint(left) and translational joint(right) [1]. . . . . 5
2.2 An anthropomorphic robot arm(left) and a spherical robot arm(right). The red arrows shows the degrees of freedom [1]. . . . . 6
2.3 Block diagram of a PID-controller with feedback [2]. . . . . 7
2.4 PWM signal with 80% Duty Cycle [2]. . . . . 10
2.5 Robot's coordinate system from above in the $x - y$ plane (drawn on Microsoft Word 2016). . . . . 11
2.6 Robot's coordinate system from sideways in the $z - xy$ plane (drawn on Microsoft Word 2016). . . . . 12

3.1 Final design . . . . . 15
3.2 Wiring for motor A and motor B with associated potentiometers . . . . 15
3.3 Wiring for motor C with associated potentiometer . . . . . 16
3.4 Homemode tightener . . . . . 16
3.5 Bearings for rotation of the base . . . . . 17
3.6 Sensor attached to servo motor . . . . . 17
3.7 Connection diagram of the system (made on Fritzing). . . . . 18
3.8 Flowchart [2]. . . . . 20
3.9 Test station . . . . . 21
3.10 Angular movement within small distance, 16-15° . . . . . 22
3.11 Angular movement within small distance, 15-12° . . . . . 22
3.12 Angular movement within medium distance, 12-7° . . . . . 23
3.13 Angular movement within medium distance, 8-0° . . . . . 23
3.14 Angular movement within large distance, 0-40° . . . . . 24
3.15 Angular movement within large distance, 0-60° . . . . . 24

# List of Tables

3.1 Start values and end values for the small distance, 16-15° . . . . . 25
3.2 Start values and end values for the small distance, 15-12° . . . . . 25
3.3 Start values and end values for the medium distance, 12-7° . . . . . 25
3.4 Start values and end values for the medium distance, 8-0° . . . . . 25
3.5 Start values and end values for the large distance, 0-40° . . . . . 26
3.6 Start values and end values for the large distance, 0-60° . . . . . 26

# Chapter 1

# Introduction

## 1.1 Background

Today's society is increasingly dependent on robotic influences in the daily life. The automation of work has for example been a big part since the industrial revolution. Thereby the assembly line was changed to a whole new efficiency of work. Many industrial companies invest in robots, because they are able to do tasks with accuracy, quickness and repeatability that outclasses the human. However robots are not only supposed to replace human tasks, there are also robots with purpose to help and assist people with reduced abilities. In this project a feeding robot was an idea of helping people with disabilities to assist in feeding them.

## 1.2 Purpose

The purpose of this project was to help feeding a person with reduced abilities. The goal was to search for food in a certain movement pattern with a distance sensor and go to a persons mouth after detecting the food. For precise movements and feedback, different PID-controllers were implemented. For this purpose the below research questions were investigated:

- What is the most appropriate design of a robot arm used for the scope of this project?

- Which controller is suitable to follow a given pattern as precisely and fast as possible?

- How does the theoretical control parameters differ from the parameters that were used on the robot?

## 1.3 Scope

Within 20 weeks a project in mechatronics is accomplished. Due to a limited time, there were some limitations. The purpose was to obtain a simple feeding motion, with a robot arm that has four joints with a total of four degrees of rotational freedom which includes the motion of the servo motor that has a sensor attached to it. The sensor used for food detection had to face down during the whole given pattern and when food was detected, the arm was supposed to stop right above the food. After finishing the food detection part, the robot moved to the given position which was in front of the persons mouth. If food was not detected, the robot finishes the whole pattern.

## 1.4 Method

A CAD model of the construction was designed in Solid Edge which made it possible to create the parts by laser cutting or 3D-printing. Most of the parts were in wooden and plastic since an abundance of those materials were available during the project.

Due to lack of datasheets for the motors, the data was experimentally performed. The torque constant $K$ was obtained by [3]

$$V = Ri + K\dot{\theta}. \tag{1.1}$$

The resistance $R$ was obtained by measuring with a multimeter. The angular velocity, $\dot{\theta}$ could be calculated by using a wire attached to the motor axle (after gearing). The voltage and current were read off a voltage source. For motor C (steers the base of the robot) the ratio was found on a limited datasheet[4]. For motor A (steers the lower arm) and motor B (steers the upper arm) which are the same type of motors, one motor was demounted to obtain the ratio. The motor axle (before gearing) was created on CAD with approximated density on the material to easily obtain the moment of inertia, $J_{rotor}$ of each motor. $J_{system}$ for each motor was also obtained on CAD.

An experiment with purpose of comparing how the system behaves with different controllers was conducted. The focus was on motor A and to investigate which controller was the most suitable for the system controlled by the motor. The experiment was divided into three main parts: smaller, medium and larger angular distances. Each main part was done with two angular distances within small and large ranges. All these angular distances corresponded to the movement of motor A during the pattern. The goal with the experiment was to compare how each tested controller performes in different intervals. The different controllers were obtained by tuning one parameter at the time. The choice of the parameters changes the behavior, therefor the parameters were modified with basis of the theory in Section 2.2.

During the movement of each test, the angle and time were frequently displayed on the Arduino environment's built-in serial monitor[5]. Data was then collected and documented in MATLAB to obtain plots of the behavior of the system. The results of each controller were then compared.

# Chapter 2

# Theory

## 2.1 Past Research

The robotic arm is a wide and well researched subject. It appears everywhere, from the medical industry as surgical tools, to systems like the ABB-robots, which are used in a wide area within the industry [6]. When it comes to large projects like these, the design and the motion of the arm are well thought out which gives a lot of inspiration. The problem is that these projects have a precision that is very high, which leads to a high cost. For that reason, these high-end robot designs are not optimal for a low budget feeding robot.

The robotic arm has become more common as lower cost hobby devices. There are plenty of forums, for example "Instructables" [7] or "Reddit" [1], that discuss different problems and solutions, which creates a big opportunity to solve all kinds of problems with a lower budget.

Usually the robot arm is designed to have three degrees of freedom. Each joint has one degree of freedom and are able to move in either a rotational or a translational motion, see Figure 2.1.

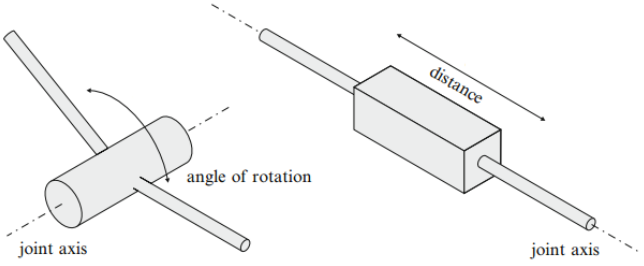



**Figure 2.1.** Rotational joint(left) and translational joint(right) [1].

With different combinations of these structures in the joints, various character-

istics of robot arms are enabled. Two common types are anthropomorphic robot arm and spherical robot arm, see Figure 2.2.

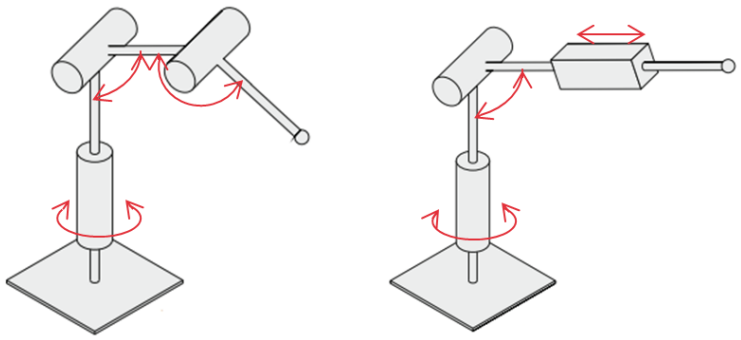


**Figure 2.2.** An anthropomorphic robot arm(left) and a spherical robot arm(right). The red arrows shows the degrees of freedom [1].

There has been previous projects of feeding robots. Assistive Innovations Corp designed a feeding robot [8] where they created a movement by a spherical arm on the robot. The Obi team also designed a feeding robot [9] but they created a movement by an anthropomorphic arm. The anthropomorphic arm has the advantage of being able to reach the vertical rotational axis, as opposed to the spherical robotic arm. The difference between the two structures is that the spherical structure can not reach things that is too close to its own body, while anthropomorphic is able to reach almost everywhere within its reach. The benefit with the ability to work close to its own body is that it could have the plate close to its body and cover less space on the table. Therefor, the decision was made to design and manufacture an anthropomorphic arm.

A common way to steer a robotic arm is either by implementing a servo motor or a stepper motor, because of their built-in ability to sense the position of the motor. While using a DC motor, a potentiometer or an encoder has to be added to measure the position. A controller is also required to obtain a precise motion [10]. By attaching servo motors or stepper motors on each axis of the robot arm, it is easier to program the desired motion. The problem with this solution is that both motors needs current constantly to keep a static position, which drains a lot from the power source.

Jonas Forsslund, Michael Yip and Eva-Lotta Sallnäs designed a robot arm using wires [11]. This makes it possible to use mechanical advantage between the motor and the arms, to transmit higher torque if needed. With wires the motors will not be limited to be fastened on the rotational axle, this results in the arms not having to care the weight of the motors. Worm geared DC motors has the characteristic that it is self inhibiting which is suitable for robot arms hence it enables the robot

to stay in a static position without any current. The dilemma with this is the need of using potentiometers to read the position of the motor and construct a control regulator to make a dynamic movement. Despite the extra work, DC motors were considered to be the best solution and was applied in this project.

## 2.2 PID-control

A method to control a system is by feedback, this means that the output signal, $y(t)$ of the system gives information for adjusting the control signal, $u(t)$, see Figure 2.3.

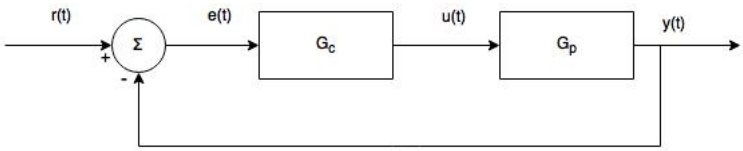



**Figure 2.3.** Block diagram of a PID-controller with feedback [2].

The error, $e(t)$ needs to converge zero to have a better performance of the control system and the control signal should be adjusted to achieve it. The difference between the output signal and the reference signal, $r(t)$ is the error value

$$e(t) = r(t) - y(t). \tag{2.1}$$

The control signal of a PID-regulator is given by

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{d}{dt} e(t) \tag{2.2}$$

where $\tau$ is the time variable of integration.

The effects on the system depends on the chosen parameters on the P (proportional), I (integrative) and D (derivative) parts of the controller. A higher proportional value gives an increase of the speed of system, but makes the system unstable, which means that the output signal oscillates. Only the use of a P-controller results in an error that is always greater than zero. By adding the integral part the errors in the output can be eliminated but the stability further decreases. A higher value on the derivative part improves the stability [12].

The transfer function of the PID-controller, $G_C(s)$ is given by

$$G_C(s) = P + \frac{I}{s} + Ds = \frac{Ps + I + Ds^2}{s}. \tag{2.3}$$

The transfer function of the plant, $G_P(s)$ is formulated in Section 2.3, where the poles were achieved by calculations of the denominator's zeros. The pole placed farthest from origin in the left half plane was chosen, since that gives a fast and stable system. When all the transfer functions of the open loop system were obtained, the closed loop function, $G_{Cl}(s)$ was achieved by

$$G_{Cl} = \frac{G_C G_P}{1 + G_C G_P}. \tag{2.4}$$

The closed loop system had three poles and the fast pole was obtained by $G_P(s)$. To find the parameters, the denominator of Equation 2.4 was used for pole placement

$$1 + G_C G_P = (s - pole)^3. \tag{2.5}$$

## 2.3 Model of the system

In this project PID-controllers were implemented. To find the transfer function of the closed loop system, a relationship in Laplace's transform was found between the output signal and the control signal. With DC motors included in the system the output signal was the angle of the motor axle $\theta(s)$ and the control signal was the voltage of the motor $V(s)$.

A relationship was formulated to find the transfer function of the system. Hence, the relationship of torque was given by [3]

$$M_{acc} = M_m \text{ - } M_{load} \tag{2.6}$$

where the torque of the acceleration is

$$M_{acc} = J_{rotor}\ddot{\theta} + b\dot{\theta} \tag{2.7}$$

Due to gear ratio the $M_{load}$ was formulated as

$$M_{load} = \frac{1}{(n_m n_{system})^2} J_{system}\ddot{\theta} \tag{2.8}$$

where $n_m$ is the gear ratio of the motor and $n_{system}$ is the gear ratio between the motor axle (after gearing) and the axle that rotates the robot.

The torque of the motor was given by [13]

$$M_m = K_t i. \tag{2.9}$$

Kirchoff's circuit law was used on a DC motor model which gave

$$V - Ri - L\frac{di}{dt} - e_{emf} = 0 \tag{2.10}$$

where the back emf, $e_{emf}$ was

$$\mathrm{e}_{emf} = \mathrm{K}_e\dot{\theta}. \tag{2.11}$$

The $K_t$ and $K_e$ are equal when SI units were considered, therefor they were replaced by a $K$ to represent both of them.

Substitution of Equation 2.7, 2.8 and 2.9 into 2.6 and isolation of the current $I(s)$ after the Laplace transform, the equation is following

$$I(s) = \frac{(J_{rotor}s^2 + bs + \frac{1}{(n_m n_{system})^2}J_{system}s^2)\theta(s)}{K}. \tag{2.12}$$

Substitution of Equation 2.11 into 2.10 and with Laplace transformation the equation gives

$$V(s) - RI(s) - LsI(s) - Ks\theta(s) = 0. \tag{2.13}$$

By inserting Equation 2.12 into 2.13 the transform function is obtained by placing $\theta(s)$ in the numerator and $V(s)$ in the denominator

$$G_P = \frac{K}{(J_{rotor}L + \frac{1}{(n_m n_{system})^2}J_{system}L)s^3 + (J_{rotor}R + \frac{1}{(n_m n_{system})^2}J_{system}R + bL)s^2 + (K^2 + bR)s}. \tag{2.14}$$

## 2.4 PWM (Pulse Width Modulation)

PWM is a method where digital sources are used to obtain analogue signals. This is a good way to simulate any different voltage between 0 V and the maximum voltage of DC motors. The digital signal is a square wave and follows an on-off pattern where the voltage switches between high state and low state [14]. The percentage of time that the signal is high during one completed cycle is called the duty cycle. Which means if voltage of 4 V is desired from a source of 5 V, then the duty cycle needs to be at 80% [15], see Figure 2.4

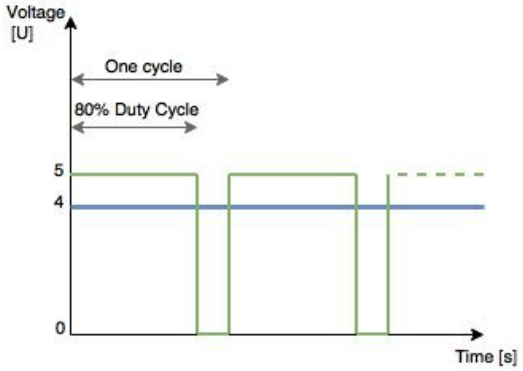



**Figure 2.4.** PWM signal with 80% Duty Cycle [2].

## 2.5 Ultrasonic sensor

An ultrasonic sensor has an ultrasonic transmitter and a receiver, where the transmitter emits a sound wave of 40 kHz which hits an object and reflects back the sound wave to the receiver. The time interval to complete the sending and receiving can be used to calculate the distance between an object and a sensor [16]

$$distance = \frac{time \cdot c}{2} \tag{2.15}$$

where $c$ is the speed of sound. Considering in air at a temperature of 20°, it is 343m/s [17].

## 2.6 Coordinate System

To measure distance in space, there was some coordinate system to refer to. Since the robot was steered by DC motors and potentiometers, the movement of the arm was measured in degrees. With the angle of each joint and the lengths of each arm, the system could be described by spherical coordinates [18].

To make it easier to measure distances and find specific coordinates, the system was described by Cartesian coordinates.

By using Cartesian coordinates there were two upcoming problems, inverse and forward kinematics[19]. When the robot received a coordinate to reach, inverse kinematics was needed to translate the Cartesian coordinates to angles that the DC motors rotated to. When the coordinates for the robot was known, forward kinematics was used to translate its angle position to Cartesian coordinates.

By using forward kinematics and trigonometry the robot's coordinates [$P_x$,$P_y$, $P_z$] were described by

$$P_x = cos(\theta_1)(L_1 sin(\phi_1) + L_2 sin(\phi_2)), \tag{2.16}$$

$$P_y = sin(\theta_1)(L_1 sin(\phi_1) + L_2 sin(\phi_2)), \tag{2.17}$$

$$P_z = L_1 sin(\phi_1) - L_2 sin(\phi_2). \tag{2.18}$$

The inverse kinematics required more calculations than forward kinematics. Since there were only two degrees of freedom in the $z-xy$ plane, see Figure 2.6 and one degree of freedom in $x-y$ plane, see Figure 2.5 the calculations were simplified to find $\theta_1$, $\phi_1$ and $\phi_2$ by using trigonometry and Pythagaros theorem [18]. In Figure 2.5, $L_1$ is the lower arm and $L_2$ is the upper arm.

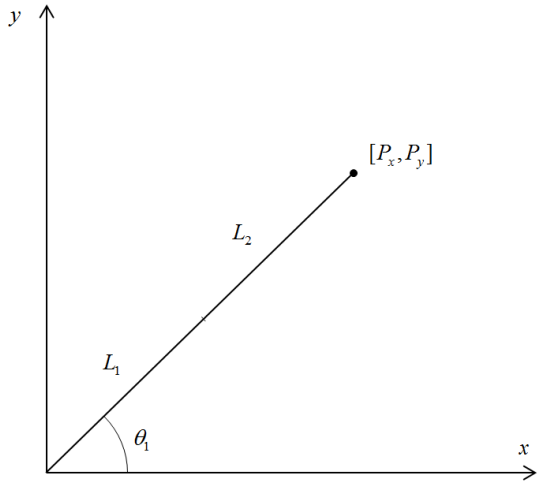



**Figure 2.5.** Robot's coordinate system from above in the $x-y$ plane (drawn on Microsoft Word 2016).

The angle, $\theta_1$ is also shown in Figure 2.5 and was given by

$$\theta_1 = \arctan(P_y/P_x). \tag{2.19}$$

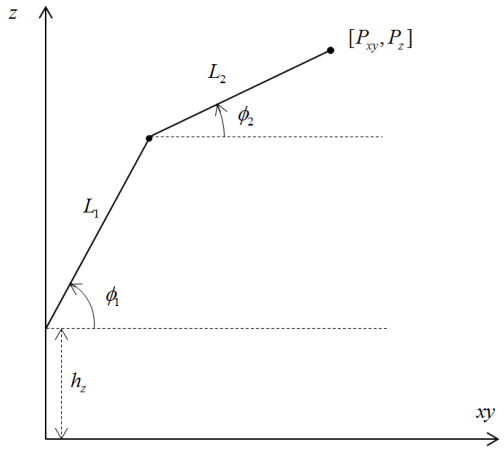



**Figure 2.6.** Robot's coordinate system from sideways in the $z - xy$ plane (drawn on Microsoft Word 2016).

In Figure 2.6, the height up to the lower arm's rotational axle, $h_z$ is shown and the coordinate $P_{xy}$ is given by

$$P_{xy} = \sqrt{P_x^2 + P_y^2}. \tag{2.20}$$

The angles $\phi_1$ and $\phi_2$ in Figure 2.6 is given by

$$\phi_1 = \arctan\Big(\frac{P_z - h_z}{\sqrt{P_x^2 + P_y^2}}\Big) + \arccos\Big(\frac{L_1^2 + P_x^2 + P_y^2 + (P_z - h_z)^2 - L_2^2}{2L_1\sqrt{P_x^2 + P_y^2 + (P_z - h_z)^2}}\Big) \tag{2.21}$$

and

$$\begin{aligned}\phi_2 = \ & \arccos\Big(\frac{L_2^2 + L_1^2 - P_x^2 - P_y^2 - (P_z - h_z)^2}{2L_1L_2}\Big) + \arctan\Big(\frac{P_z - h_z}{\sqrt{P_x^2 + P_y^2}}\Big) \\ & + \arccos\Big(\frac{L_1^2 + P_x^2 + P_y^2 + (P_z - h_z)^2 - L_2^2}{2L_1\sqrt{P_x^2 + P_y^2 + (P_z - h_z)^2}}\Big) - 180^\circ.\end{aligned} \tag{2.22}$$

# Chapter 3

# Demonstrator

## 3.1 Problem formulation

The main parts of the project were to find suitable controllers for this construction, to be able to track the trajectory and detect the food with the sensor, to feed the person. Thus, there were some requirements that needed to be fulfilled.

- Ability to steer the movement with potentiometers, DC motors and implemented controllers
- Detect the food in a given area
- Ability to reach for a point to a person's mouth where the person must have an average height
- Stable system
- Overshoot should be avoided
- Reach the desired point as fast as possible
- Should not exceed the acceptable error

## 3.2 Hardware

### 3.2.1 Chosen parts

Electronic components used in the prototype:

- 1 DC-motor (DME25BA–005)
- 2 DC-motors (CSD71P1)
- 2 H-bridges (L298)
- 3 Potentiometers
- Ultrasonic sensor (HC-SR04)
- Servo motor (SG90)

Motor A and motor B are larger, heavier and more powerful than motor C. Motor A and motor B were placed on the base of the robot and wires were used to transfer the movement to the arm, this to prevent an unnecessarily heavy robot arm. Motor A and B steers the robot arm, while motor C steers the rotation of the base of the robot which motor A and B are grounded on. All motors have worm gears to obtain a self-inhibitory effect, because it was desirable for the arm to be steady and hold up without current. The strength of a few motors were tested before deciding which one to implement, with the goal to be able to lift a weight as much as a bite of food.

Two H-Bridges were required for the use of three motors, since one H-bridge was used to control two motors. With a H-bridge it allows the controlling of motors in both directions. The components were fastened similar to a 2A Dual L298 H-bridge. The H-bridge was chosen to be made by hand, since all the components were available in the laboratory.

The use of potentiometers with a resistance of 4.7 k$\Omega$ was to control the motors by tracking the arm movement. A potentiometer is like an adjustable resistor, which makes the output voltage variable depending on the position of the wiper [20]. The analogue potentiometers were attached with wires on the construction. Two potentiometers were for motor A and motor B and the third was for motor C.

A sensor which could detect a distance difference was desired, therefor an ultrasonic sensor was chosen. The minimum range for the chosen sensor was 2 cm [16] which was suitable for this project. The sensor was attached to the servo motor on the end of the arm, which allowed the sensor to always face down when detecting food. The servo motor was chosen due to its light weight.

### 3.2.2 Design

The main parts of the prototype were constructed in wood and were obtained by laser cutting. The arms were laser cut with cavities to avoid unnecessary weight. Some components were made of 3D-printed PLA plastic. The two arms on the robot were controlled by wires from the DC motors and potentiometers. The final design of the robot is shown in Figure 3.1.

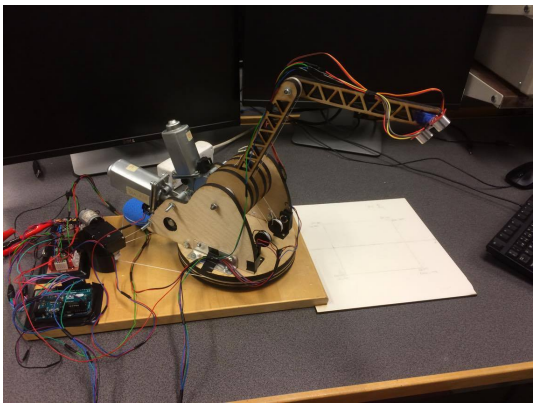


**Figure 3.1.** Final design

The holders of all the potentiometers were 3D-printed. To avoid tangles and with the lack of space, motor A, B and associated potentiometers were placed on each side of the robot base. These potentiometers were fasten by screws and 3D-printed support. The wires were attached on the same axle but different holders on the robot, see Figure 3.2

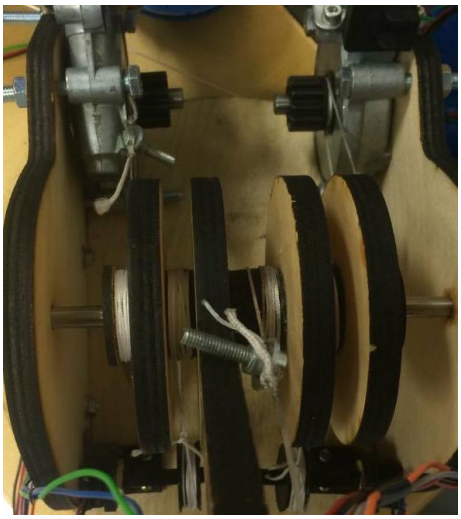


**Figure 3.2.** Wiring for motor A and motor B with associated potentiometers

A holder was 3D-printed for motor C, to hold both wires from the potentiometer on the upper part and the wires from the rotational base of the robot on the lower part. A support for the motor C was also 3D-printed to adapt the motor shaft to the right height, see Figure 3.3

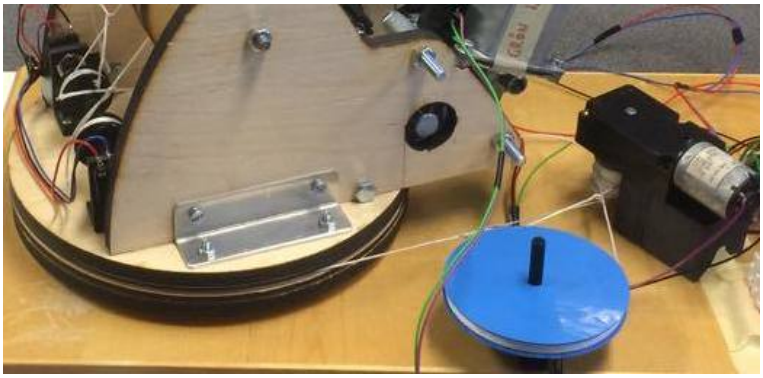


**Figure 3.3.** Wiring for motor C with associated potentiometer

The wires that steers the robot arm is connected with small screws and metal washer to obtain enough tension, see Figure 3.4. The screw and metal washer has its function to tighten the wires. The wires on the potentiometer and the rotational base does not need as much tension and therefor the homemade tightener was not attached there.

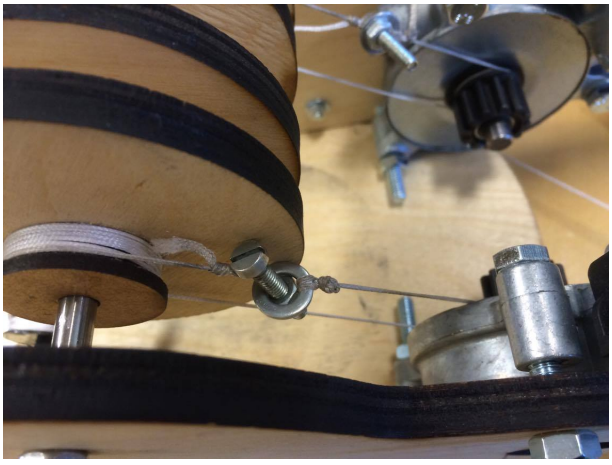


**Figure 3.4.** Homemode tightener

The base of the robot allowed rotation by eight glass balls. To ensure that the glass balls were kept in place and able to rotate, there was a wooden plate with holes and small wooden supports under the plate for each glass ball, see Figure 3.5

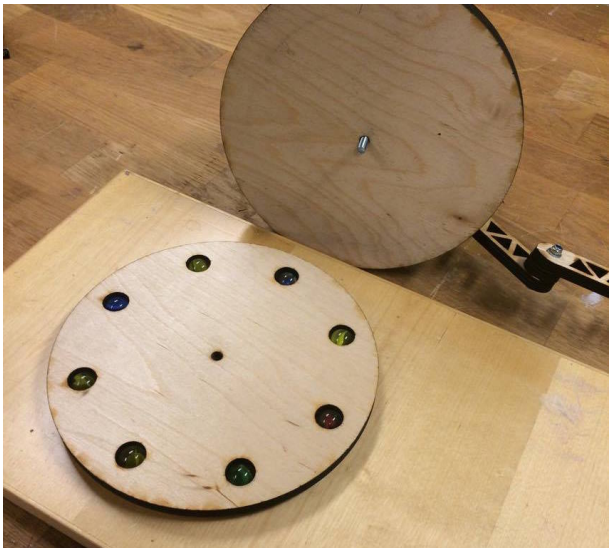


**Figure 3.5.** Bearings for rotation of the base

The servo was fasten by cutting a hole in the arm and pushed in place, the sensor was glued to the servo motor, see Figure 3.6

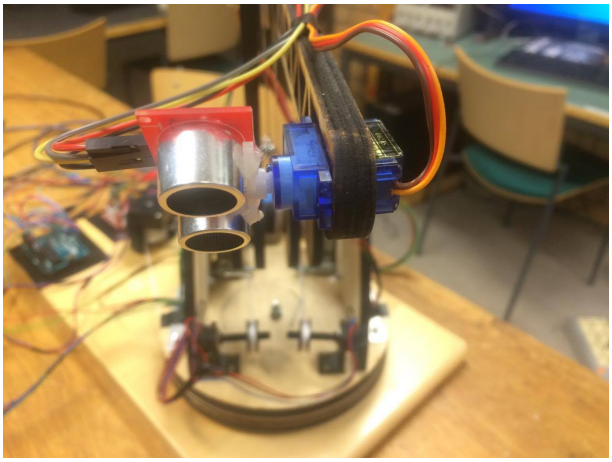


**Figure 3.6.** Sensor attached to servo motor

## 3.3 Electronics

The entire system consists of a microcontroller, three DC motors, three potentiometers, two H-bridges, one servo motor and one ultrasonic sensor. The H-bridges connects the DC motors to the Arduino Uno. The movement of the robot is run by an Arduino Uno, which has a microcontroller on the board. Arduino reads inputs and converts it to outputs. By uploading a code in Arduino programming language to the microcontroller, it is possible to give the Arudino board instructions [21]. The analogue potentiometers are connected directly to the analogue inputs of the Arduino, through the software the microcontroller can convert analogue values to digital values. The potentiometers are also connected to Arduino's 5 V and GND, this means the input will be in a range between 0 V and 5 V, which will be converted to a digital value between 0 and 1023. The sensor is connected to GND, 5 V and has an Echo pin and a Trig pin that connects to the digital inputs. The servo motor is connected to GND, 5 V and a digital input.

Figure 3.7 demonstrates the connection diagram of the system, with the only difference that the external source was replaced with batteries.

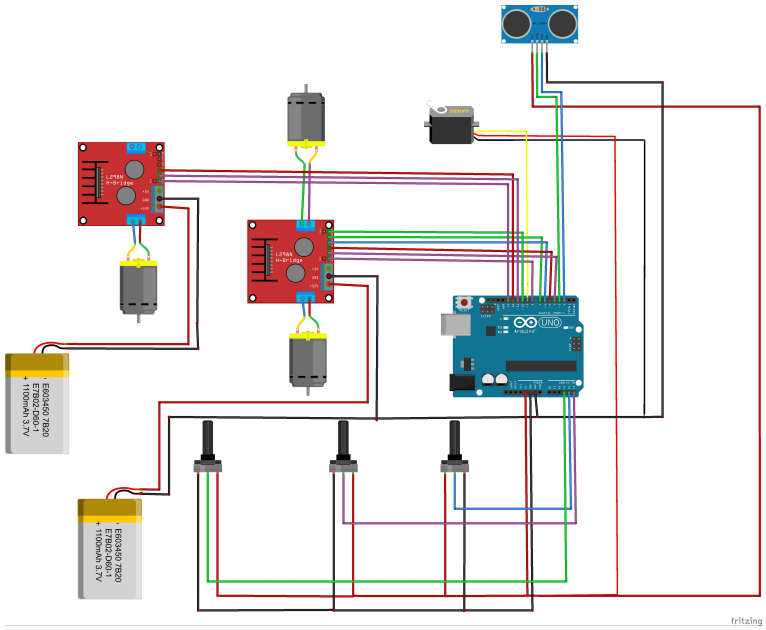


**Figure 3.7.** Connection diagram of the system (made on Fritzing).

Motor A and motor B were connected to external source of 12 V and a maximum current of 1.9 A and motor C was powered with 12 V and a maximum current of 0.5 A.

## 3.4 Software

The Arduino program reads the potentiometer as analogue values, therefor the desired angle was also converted to analogue values. The difference between these values was continuously calculated in a loop until an acceptable error was reached, thereby the motor was turned off. This error gave directions to the DC motor. By using the built-in PID library in the Arduino program, a PWM-signal was calculated and regulated through the loop, to be sent to the motor. The robot arm was steered with one motor at a time through the given pattern. The ultrasonic sensor was driven only when motor C was running. If the sensor detected a change in distance, an obstacle was found, thereby the loop was broken and the robot arm stopped above the obstacle, to later move to a given position of an illustrated mouth. The angle for the servo motor was only depending on the angle of motor B's output shaft and was controlled by using Arduino's built-in servo library. The given angle for the servo motor corresponded to the same angle of the potentiometer connected to motor B. Figure 3.8 describes a flowchart of the Arduino code.

The movement pattern was obtained from the MATLAB code, see Appendix C, by transforming coordinates of the pattern to angles for the motors. These angles were then used in the Arduino code.

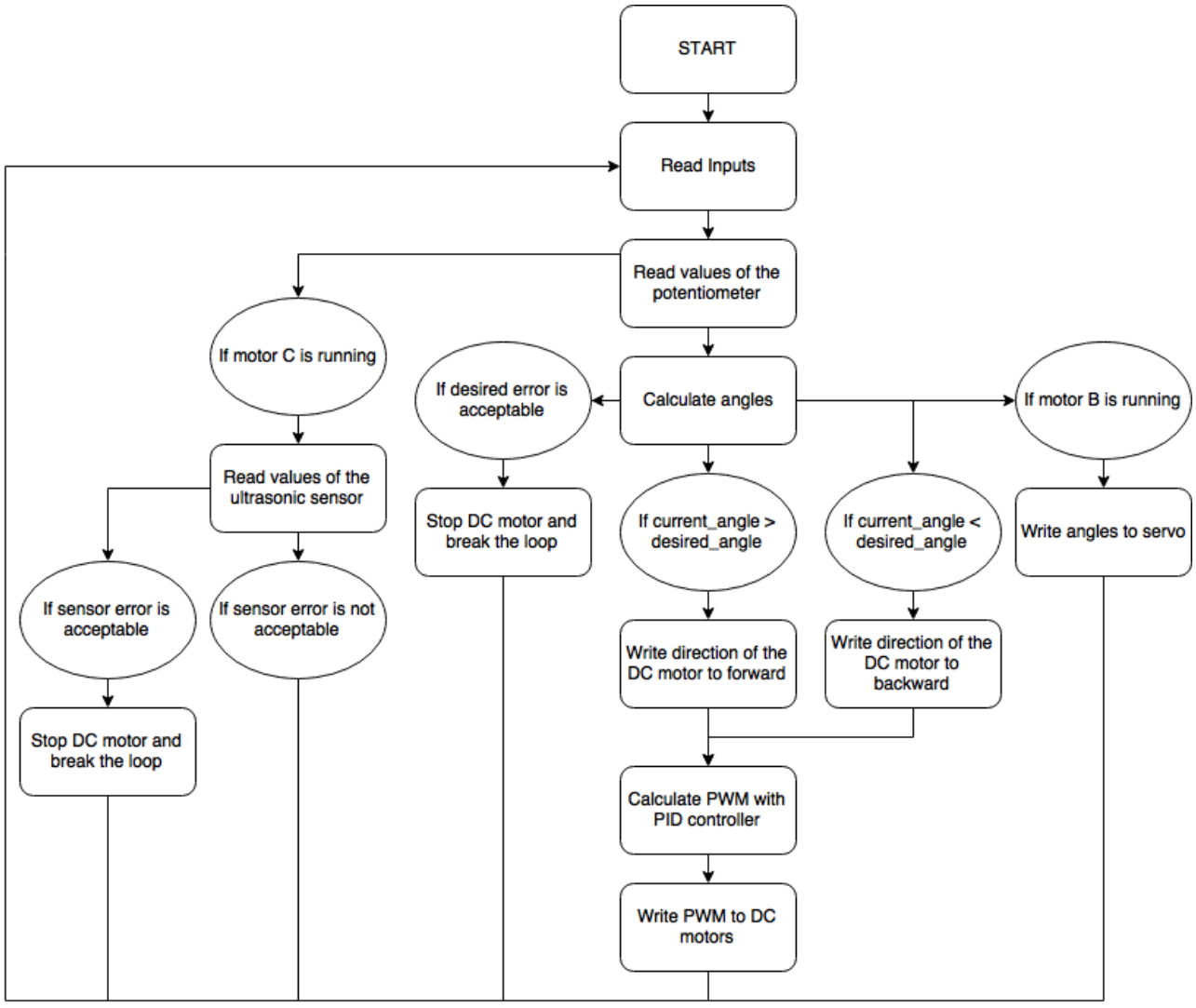

START
Read Inputs
Read values of the potentiometer
If motor C is running
Read values of the ultrasonic sensor
If sensor error is acceptable
If sensor error is not acceptable
Stop DC motor and break the loop
If desired error is acceptable
Stop DC motor and break the loop
Calculate angles
If current_angle > desired_angle
Write direction of the DC motor to forward
If current_angle < desired_angle
Write direction of the DC motor to backward
If motor B is running
Write angles to servo
Calculate PWM with PID controller
Write PWM to DC motors

**Figure 3.8.** Flowchart [2].

## 3.5 Results

The experiment of testing different controllers on motor A was made without the sensor and the servo motor attached to the construction, see Figure 3.9.

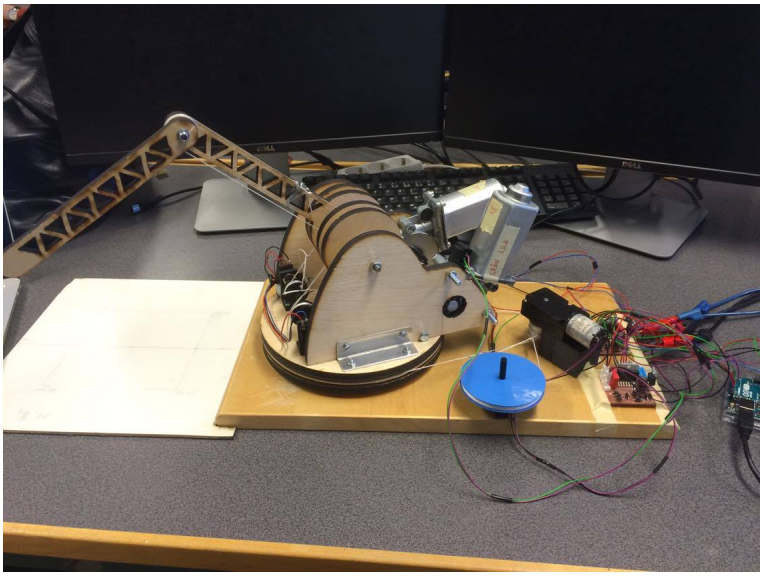


**Figure 3.9.** Test station

The results of testing P-, PI-, PD- and PID-controller applied on motor A were plotted on MATLAB, see Figures 3.10 - 3.15. The error was set to 1° on the experiment.

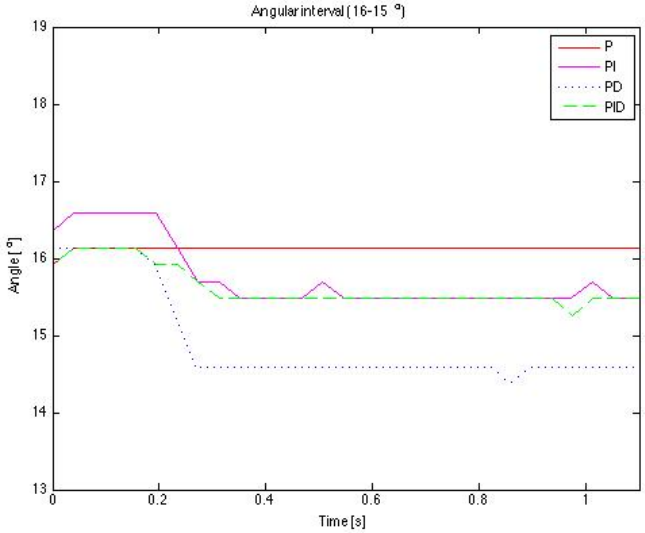



**Figure 3.10.** Angular movement within small distance, 16-15°

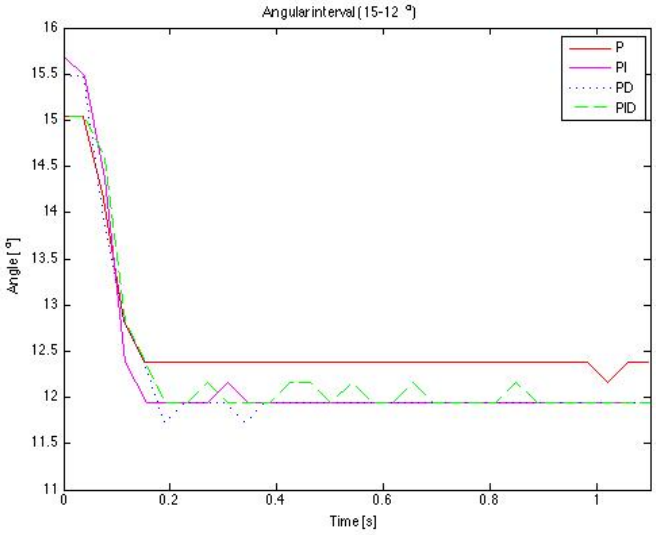



**Figure 3.11.** Angular movement within small distance, 15-12°

Figure 3.10 and 3.11 has following values for P-controller: $K_P = 12$, PI-controller: $K_P = 12$, $K_I = 4.2$, PD-controller: $K_P = 14$, $K_D = 5$ and PID-controller: $K_P = 14$, $K_I = 4.5$, $K_D = 4.5$.

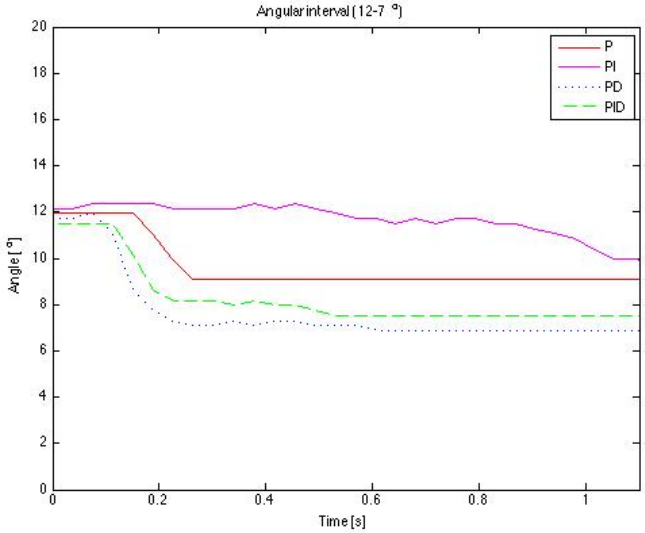



**Figure 3.12.** Angular movement within medium distance, 12-7°

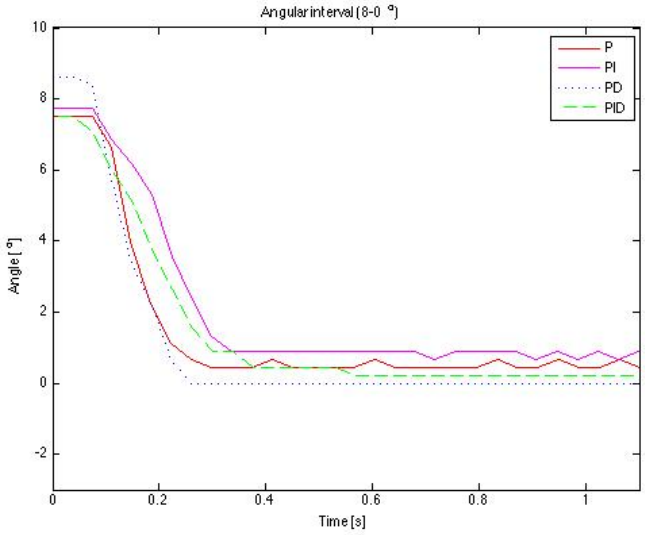



**Figure 3.13.** Angular movement within medium distance, 8-0°

Figure 3.12 and 3.13 has following values for P-controller: $K_P = 3.9$, PI-controller: $K_P = 4$, $K_I = 1.3$, PD-controller: $K_P = 5$, $K_D = 0.3$ and PID-controller: $K_P = 5$, $K_I = 0.8$, $K_D = 0.3$.

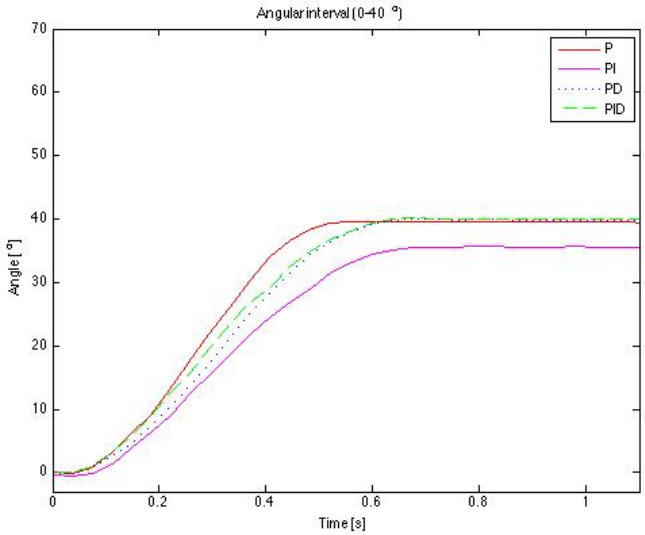



**Figure 3.14.** Angular movement within large distance, 0-40°

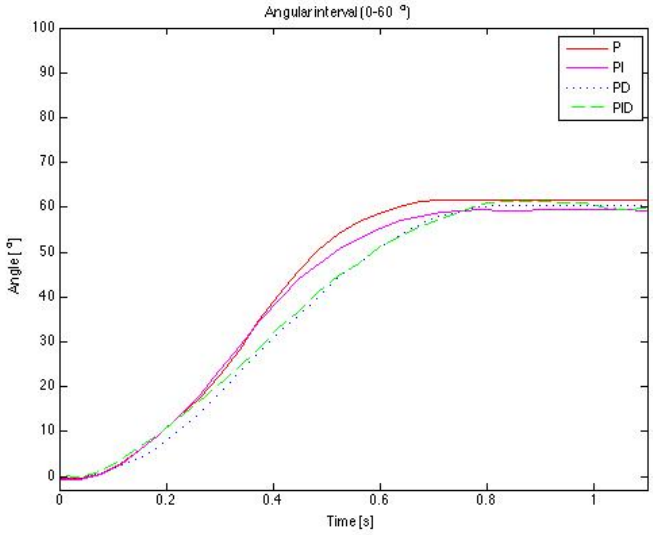



**Figure 3.15.** Angular movement within large distance, 0-60°

Figure 3.14 and 3.15 has following values for P-controller: $K_P = 1.6$, PI-controller: $K_P = 0.9$, $K_I = 0.5$, PD-controller: $K_P = 3.5$, $K_D = 0.6$ and PID-controller: $K_P = 2.2$, $K_I = 0.5$, $K_D = 0.55$.

The start values, end values and the time for reaching the endpoints were inserted in tables for each test, see Tables 3.1 - 3.6.

| Controller | Start value [°] | End value [°] | Time [s] |
|---|---|---|---|
| P | 15.92 | 16.14 | 0.039 |
| PI | 16.36 | 15.48 | 0.351 |
| PD | 16.14 | 14.59 | 0.268 |
| PID | 15.92 | 15.48 | 0.312 |

**Table 3.1.** Start values and end values for the small distance, 16-15°

| Controller | Startvalue | Endvalue | Time |
|---|---|---|---|
| P | 15.04 | 12.38 | 0.152 |
| PI | 15.70 | 11.94 | 0.155 |
| PD | 15.48 | 11.94 | 0.227 |
| PID | 15.04 | 11.94 | 0.694 |

**Table 3.2.** Start values and end values for the small distance, 15-12°

| Controller | Start value | End value | Time |
|---|---|---|---|
| P | 11.94 | 9.07 | 0.262 |
| PI | 12.16 | 8.85 | 2.682 |
| PD | 11.72 | 6.86 | 0.61 |
| PID | 11.50 | 7.52 | 0.532 |

**Table 3.3.** Start values and end values for the medium distance, 12-7°

| Controller | Startvalue | Endvalue | Time |
|---|---|---|---|
| P | 7.52 | 0.44 | 0.298 |
| PI | 7.74 | 0.88 | 0.337 |
| PD | 8.62 | 0.00 | 0.259 |
| PID | 7.52 | 0.22 | 0.57 |

**Table 3.4.** Start values and end values for the medium distance, 8-0°

| Controller | Startvalue | Endvalue | Time |
|---|---|---|---|
| P | 0.00 | 39.58 | 0.556 |
| PI | -0.22 | 35.60 | 0.675 |
| PD | -0.22 | 39.80 | 0.876 |
| PID | 0.00 | 40.02 | 0.718 |

**Table 3.5.** Start values and end values for the large distance, 0-40°

| Controller | Startvalue | Endvalue | Time |
|---|---|---|---|
| P | -0.44 | 61.47 | 0.712 |
| PI | -0.66 | 59.48 | 0.905 |
| PD | -0.44 | 60.37 | 0.824 |
| PID | 0.22 | 60.15 | 1.292 |

**Table 3.6.** Start values and end values for the large distance, 0-60°

The theoretical calculations were based on Section 2.2 and 2.3. The results for motor A with the fastest pole at -3.6652 has the parameters: $K_I = 1.6331$, $K_D = 0.3647$ and $K_P = 1.3367$.

# Chapter 4

# Discussion and conclusion

## 4.1 Discussion

Most of the errors on the system was because plastic wires were used. Because plastic is an elastic material, the wires would always get stretched. This would not be a problem if metal wires were used instead. The error was not only affected by the elasticity of the wires but also the quality of the tied knot either with or without the homemade tightener, see Figure 3.4. If the knot was not tied well enough it would slip, therefor the knots were tied in a way to ensure it would not slip too much. The ends were burned to create some resistance if the knot would slip. It was important that the wires were wrapped around with enough laps to prevent the wires to slip and not rotate the axle. The wrapping needed to be strong enough, this to ensure that the wires would not create a kind of delay on the motion. If the wires were wrapped too loose, then a time delay would occur until the axle started to rotate. Bad wiring also affected the installation of the system, hence it had to be redone which was very time consuming. The potentiometers were limited to turn less than a lap, which made it important to not exceed this interval otherwise the consequence would be broken potentiometers. When testing the robot there was a few times when it moved to undesired positions and the power source was turned off quickly to prevent the potentiometers to break. After each failed time, the wiring also had to be redone.

Some problems occurred associated with the ultrasonic sensor. One problem that was noticed was that the sensor reacted better for longer distances between sensor and object. This was not a benefit for this project, hence the distance between the sensor, surface and object was small. Another problem was that the servo motor did not move precisely with respect to the reference angles which could create inaccurate values.

In the experiment the plots showed that the system needed different controllers depending on what motion the robot followed. In the shorter angular intervals it was

clear that the controller needed greater controller parameters. Greater controller parameters will result in a faster system with a higher PWM signal. Using the same parameters for larger angular distances as well, it would probably give a large overshoot or even maybe an unstable system. Hence, using the same controller for a small and a large angular motion will not be optimal. As it is shown in the results from the experiment in Section 3.5, one controller could not be perfect for all motion intervals.

The P-controller was often very fast and precise on one of two motions in a motion interval. The other motion in the interval often ended up with a static error. The PI-controller did not work much better with the static error than the P-controller on the other tests in the interval. Since the tests only lasted for a maximum of a couple of seconds, the $K_I$ parameter of the controller did not have time enough to eliminate the static error. If waiting a longer time for the controller to set, the error would probably be eliminated, but a slow motion like that is not optimal for the purpose of this project. The PD-controller was the one that worked the best on the tests. It was fast and accurate on all the tests. The controller gave some small overshoot, but it was never exceeding the acceptable error of one degree. Because of the $K_D$ part in the controller, the overshoot was reduced and a higher $K_P$ value could be chosen, which would make the system faster. The PID-controller also worked very well. It was not always as fast as the other controllers, but it showed an accurate and trustworthy controller. Because of both $K_I$ and $K_D$ it could be both fast and reduce overshoots, but if there was some error, the $K_I$ part was able to eliminate that error.

By only using a movement pattern with the same motion distances and one controller for each motion, it would fit perfectly with a P-controller. The P-controller was the fastest and if it just needed to be adapted for one motion, it would be the perfect choice. The motion of the robot was on the other hand not just pre-defined, it was also supposed to stop where it found food and make the movement from that position to another given position. Depending on where it sensed food, the next movement would differ, and that was why the P-controller was not appropriate. The PI-controller turned to be too slow from the tests. Therefor the question is to use a PD-controller or a PID-controller. Since the PID-controller had the $K_I$ controller parameter, it was able to eliminate the static error after a while if the position was not in the desired position. This was more preferred for a feeding robot rather than just be fast but not always that precise.

As for the overshoot it did not perform as expected. Since the robot was programmed to change direction when an overshoot was obtained, it would not behave as the theory explains it to. The program could easily be adjusted by disabling that function and let the controller do all the work. This would on the other hand risk the robot to break a lot of times while tuning different controllers, which was not an option.

The theoretical control parameters differed from the real parameters used on the robot, which was expected. One of the reasons of why this differed could be that the theoretical model was not exactly as the real model. Due to missing datasheets the torque constant was obtained by an experiment and the moment of inertia was created on CAD with approximated measurements and values. This led to approximated values on the motors. Friction forces and inductance, $L$ was also neglected. This results in imprecise values, which affects the calculations to obtain the PID-parameters. But these parameters were useful as initial parameters during the experiment of different controllers, even if the real parameters were tuned to other values in the end. All the calculated parameters were within the different values of the experimental results of PID-controller, which could be a reasonable theoretical model after all.

## 4.2 Conclusions

The use of plastic wires did fulfill the goals for this projects, but other projects may need a greater accuracy and want to avoid time consuming installation. A wire of plastic is not recommended due to the problems explained in Section 4.1.

After conclusions from the experiment, a PID-controller was suitable to follow a given pattern as precisely and fast as possible. Under Section 3.5 the figures shows that a PD-controller was also a feasible alternative. The conclusion of the theoretical control parameters was that they were in the bounds of the experimental control parameters for PID-controller.

Due to neglected parameters, approximations during the CAD creation of the construction and missing datasheets, the model of system could be affected. That could be the reasons why the theoretical control parameters differed from the parameters that were actually used on the robot.

However the robot did accomplish the goals to follow a given pattern as precisely as possible and detect an object.

# Chapter 5

# Recommendations and future work

## 5.1 Recommendations

A recommendation is to invest in a better servo motor with a higher precision of going to the desired angle, this would ensure the sensor to always face down. Another solution for the software should be found, to reduce the measurement errors of the sensor. This would improve the accuracy of the sensor. Find better potentiometer might be good too. Wires of steel should be considering instead of plastic wires, to achieve higher accuracy and to save more time.

## 5.2 Future work

A gripper is something that could be added to apply this construction in reality. If not a gripper is wanted, then a spoon could be added. If a spoon is used, then a plate would need to be created, that would have edges to enable the spoon to pick up food. Another suggestion for possible future work that could be investigated is a face detecting function, where the robot would search for a mouth instead of going to a specific given position that illustrates the height of a person's mouth.

# Bibliography


[1] T. Bajd, M. Mihelj, J. Lenarcic, A. Stanovnik, and M. Munih, *Robotics*. Springer, 2010.

[2] [cited 2016-05-10]. [Online]. Available: https://www.draw.io

[3] H. Johansson, *Elektroteknik*. KTH, 2013.

[4] A. AB, "Dc motor med snäckväxel," [cited 2016-05-05]. [Online]. Available: http://images.compotech.se.webbhotell.org/DME.pdf

[5] Arduino, "Serialmonitor," [cited 2016-05-05]. [Online]. Available: https://www.arduino.cc/en/reference/serial

[6] ABB, "Product selector," [cited 2016-05-02]. [Online]. Available: http://new.abb.com/products/robotics/product-selector

[7] Instructables, "Technology," [cited 2016-05-17]. [Online]. Available: http://www.instructables.com/tag/type-id/category-technology/

[8] A. I. Corp, "ieat," [cited 2016-04-24]. [Online]. Available: https://www.assistive-innovations.com/en/eatingdevices/ieat-feeding-robot

[9] Obi, "Obi," [cited 2016-04-24]. [Online]. Available: https://meetobi.com

[10] P. Press, *DC Motors, Speed Controls, Servo Systems*. Kent, GBR: Elsevier Science, 1972.

[11] J. Forsslund, M. Yip, and E.-L. Sallnäs, "A starting kit for crafting force-reflecting spatial haptic devices," *WoodenHaptics*, 2015.

[12] T. Glad and L. Ljung, *Reglerteknik : grundläggande teori*. Studentlitteratur AB, 2006.

[13] C. T. for MATLAB and Simulink, "Dc motor speed: Simulink modeling," [cited 2016-03-15]. [Online]. Available: http://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SimulinkModeling

[14] Arduino, "Pwm," [cited 2016-03-20]. [Online]. Available: https://www.arduino.cc/en/Tutorial/PWM

[15] N. Instruments, "What is a pulse width modulation (pwm) signal and what is it used for?" [cited 2016-03-20]. [Online]. Available: http://digital.ni.com/public.nsf/allkb/2AC662081E01E1AE86257F5D00011B6D

[16] E. Freaks, "Ultrasonic ranging module hc - sr04," [cited 2016-05-17]. [Online]. Available: http://users.ece.utexas.edu/~valvano/Datasheets/HCSR04b.pdf

[17] U. of Florida, "Sound: Propagation," [cited 2016-05-20]. [Online]. Available: http://www.phys.ufl.edu/courses/phy2053/spring14/lectures/PHY2053_04-15-14.pdf

[18] L. Råde and B. Westergren, *Mathematics Handbook - for Science and Engineering*. Studentlitteratur AB, 2003.

[19] J. Awrejcewicz, *Classical Mechanics*. Springer, 2012.

[20] resistorguide.com, "Potentiometer," [cited 2016-03-18]. [Online]. Available: http://www.resistorguide.com/potentiometer/

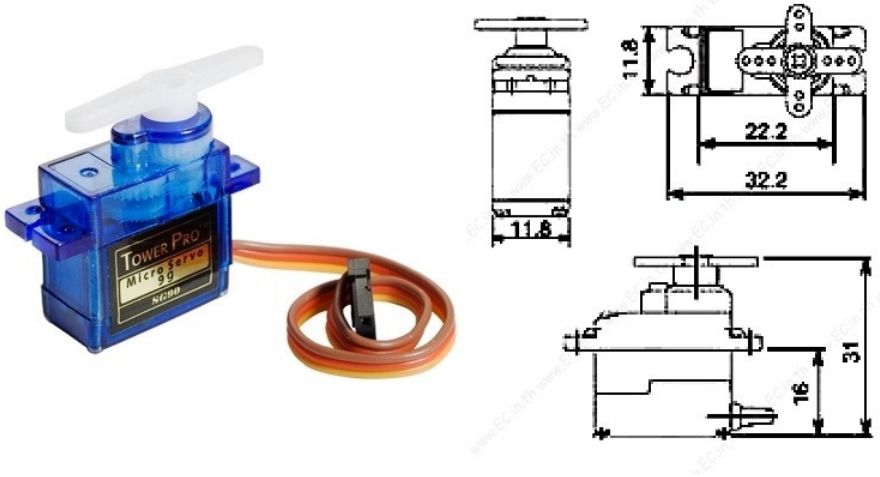
[21] Arduino, "What is arduino?" [cited 2016-03-18]. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction

# Appendix A

# Datasheets

## Datasheet for servo motor

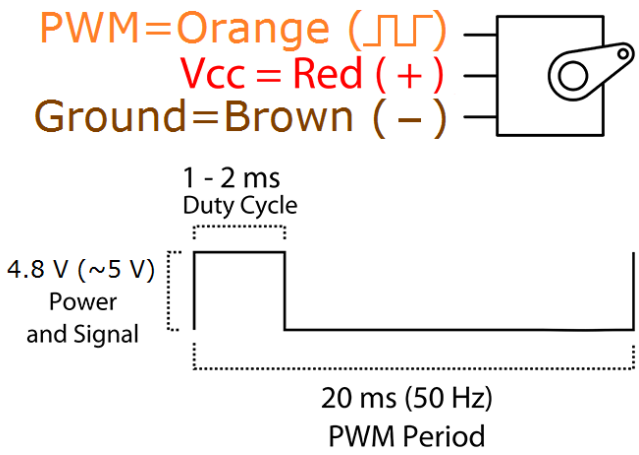
# SG90 9 g Micro Servo




Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but *smaller*. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

**Specifications**

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 1.8 kgf·cm
- Operating speed: 0.1 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 10 µs
- Temperature range: 0 ºC – 55 ºC



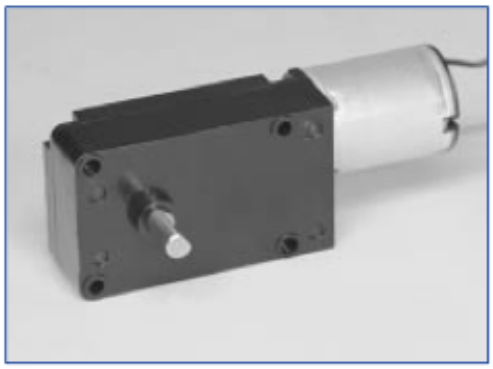

Position "0" (1.5 ms pulse) is middle, "90" (~2 ms pulse) is all the way to the right, "-90" (~1 ms pulse) is all the way to the left.

## Datasheet for motor C

### DC-motor med snäckväxel

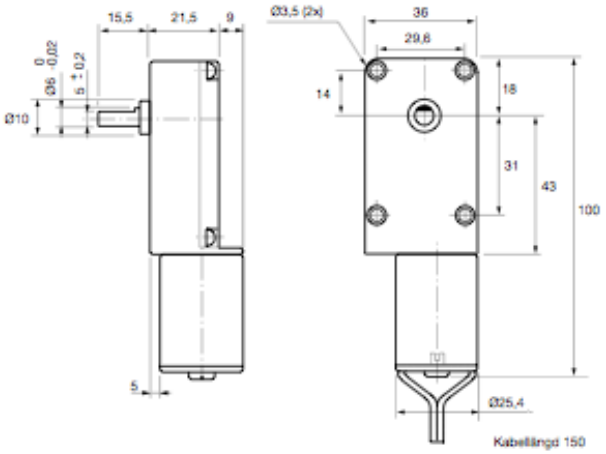
**PM-motor fabrikat Japan Servo, DME25BA-003**

- Spänning 12 och 24 V DC.
- Variabel drivspänning medger varvtalsreglering.
- Motorn är reversibel genom polvändning.
- Angivna data gäller vid korttidsdrift S3-50% (10 min).
- Varvalsreglering, se nedan.



**Motor/växel – data och beställningsnummer**

| Best. nr | Typ | Spänning V | Uteffekt W | Varvtal $min^{-1}$ | Moment Ncm | Ström mA | Startmoment Ncm | Utväxling | Vikt g |
|---|---|---|---|---|---|---|---|---|---|
| 45-155-1000 | DME25BA-005 | 12 | 1,6 | 81 | 20 | 560 | 105 | 78,9:1 | 91 |
| 45-155-1010 | DME25BA-003 | 24 | | | | 280 | | | |



# Datasheet for sensor

Tech Support: services@elecfreaks.com

# Ultrasonic Ranging Module HC - SR04

## Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

## Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

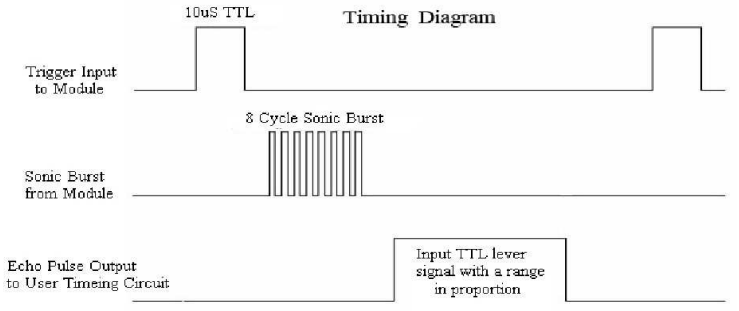
## Electric Parameter

| | |
|---|---|
| **Working Voltage** | **DC 5 V** |
| **Working Current** | **15mA** |
| **Working Frequency** | **40Hz** |
| **Max Range** | **4m** |
| **Min Range** | **2cm** |
| **MeasuringAngle** | **15 degree** |
| **Trigger Input Signal** | **10uS TTL pulse** |
| **Echo Output Signal** | **Input TTL lever signal and the range in proportion** |
| **Dimension** | **45*20*15mm** |

**Vcc Trig Echo GND**

## Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



## Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

**www.Elecfreaks.com**

# Appendix B

# Arduino Code

## Arduino Code

```
1  // Feeding Robot
2  // Filip Malberg & Linda Truong
3  // TRITA MMK 2017:26 MDAB 644
4  // KTH, ITM, MDA
5
6  // Arduino code to control angle of a motor shaft using a
       potentiometer for feedback
7  // also controlling a servo and a sensor to measure
       difference in distance
8  // The basics of the code was taken from:
9  // https://bayesianadventures.wordpress.com/2014/01/04/diy-
       servo-motor-cheap-and-with-plenty-muscle/
10
11 // Including built-in libraries
12 #include <Servo.h>
13
14 #include <PID_v1.h>
15
16 // We measuring the coordinate system using the 90 degree
       interval
17 #define MAX_ANGLE 90
18
19 // Now we define a class that will control the DC motors
20 // This motors must be running from a H bridge like L298
21 class DCMotor
22 {
23 private:
24     // Initiate motor pins
25     int M_pin1 , M_pin2, M_PWMPin;
26
27     int turnDirection;
28
29     enum turnDirection {right, left};
30
31 public:
32
33     DCMotor(int p1, int p2, int p3) //Constructor
34     {
35             M_pin1 = p1; // Direction pin on L298
36
37             M_pin2 = p2;  // Direction pin on L298
38
```

```
39              M_PWMPin = p3;   // PWM pin on L298
40
41              // Set the pins as OUTPUT
42              pinMode(M_pin1, OUTPUT);
43
44              pinMode(M_pin2, OUTPUT);
45      }
46
47      // Setting turn directions on L298
48      void SetTurnDirection(int dir)
49      {
50          turnDirection = dir;
51
52          if(turnDirection==right)
53          {
54
55                      // Motor moves up
56                      digitalWrite(M_pin1, HIGH);
57                      digitalWrite(M_pin2, LOW);
58
59          }
60          else if(turnDirection == left)
61          {
62
63                      // Motor moves down
64                      digitalWrite(M_pin1, LOW);
65                      digitalWrite(M_pin2, HIGH);
66
67          }
68      }
69
70      // Stops the motor
71      void Stop()
72      {
73          analogWrite(M_PWMPin, 0);
74      }
75
76
77      // GoToAngle is where the motors and sensors are
            controlled
78      int GoToAngle(double target, double potPin, double
           POT_VALUE_MAX, double POT_VALUE_MIN, double Kp,
            double Ki, double Kd, int sensorStart = 0)
79      {
```

```
80          double PERM_ERROR = 1; // Acceptable error
81
82          // Define Variables we'll be using in the PID
83          double Setpoint, Input;
84          double Output = 0;
85
86          // Constructing counters
87          // Starts the counting at 100
88          int counter = 100;
89          // Starts the counting at 100
90          int counterSensor = 100;
91          // time_t is a constant 1 or 0
92          int time_t = 0;
93          // Variables that will measure the time
94          unsigned long currenttime = 0;
95          unsigned long firsttime = 0;
96
97          // Defines variables of the sensor
98          long duration_curr = 0;
99          long duration_prev = duration_curr;
100         long sensorError = 40;
101         const int trigPin = 3;
102         const int echoPin = 2;
103
104         // Sets the trigPin as an Output
105         pinMode(trigPin, OUTPUT);
106         // Sets the echoPin as an Input
107         pinMode(echoPin, INPUT);
108
109         // Create an instance for the PID
110         PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd,
                DIRECT);
111         myPID.SetSampleTime(1);
112         // Initialize the variables we're using in the PID
113         Input = analogRead(potPin);
114         // Changing the target to analog values
115         Setpoint = target*(POT_VALUE_MAX - POT_VALUE_MIN)/
                MAX_ANGLE + POT_VALUE_MIN;
116
117         // Turn the PID on
118         myPID.SetMode(AUTOMATIC);
119
120         // Calculates the current angle of the motor shaft
121         double currentAngle = ((double)analogRead(potPin) -
```

```
      POT_VALUE_MIN)/(POT_VALUE_MAX - POT_VALUE_MIN) *
      MAX_ANGLE;
122
123       //First Check if we need to turn left or right
124       if (currentAngle < target)
125       {
126         SetTurnDirection(right);
127         // DIRECT or REVERSE switch the direction of the
              built-in PID library
128         myPID.SetControllerDirection(DIRECT);
129       }
130       else if (currentAngle > target)
131       {
132         SetTurnDirection(left);
133         myPID.SetControllerDirection(REVERSE);
134
135       }
136
137       // Runs the loop until it breaks
138       while(true)
139       {
140
141         // Input is the currentAngle
142         // Output is the pwm signal that myPID is going to
              calculate
143         // Setpoint is the target
144         // When the error is accaptable the loop will end
145
146         // Input is the value of the potentiometer
147         Input = analogRead(potPin);
148
149         // Computes the new Output
150         myPID.Compute();
151         // Set the new Output as the new PWM signal to the
              motor
152         analogWrite(M_PWMPin, Output);
153
154         // The counter is because the sensor need small
              breaks
155         // The sensor is only working while motor C is on
              and then sensorStar=1
156         if(sensorStart == 1 && counterSensor == 100)
157         {
158           // Clears the trigPin
```

```
159             digitalWrite(trigPin, LOW);
160             delayMicroseconds(2);
161             // Sets the trigPin on HIGH state for 10 micro
                   seconds
162             digitalWrite(trigPin, HIGH);
163             delayMicroseconds(10);
164             digitalWrite(trigPin, LOW);
165             // Reads the echoPin, returns the sound wave
                   travel time in microseconds
166             duration_curr = pulseIn(echoPin, HIGH);
167
168             // The conditions when the sensor detect a
                   difference in distance
169             // A median filter would be a appropriate
                   replacement and a better solution
170             if (duration_prev - duration_curr > sensorError &&
                    duration_curr < 600 && duration_prev < 630 &&
                   duration_prev > 570  && duration_curr > 400)
171             {
172               Stop();
173               // Returns true so the pattern in the main
                     program is stopped
174               return true;
175             }
176
177             // Set the current duration to the previous
178             duration_prev = duration_curr;
179             // Restart the counter
180             counterSensor = 0;
181           }
182           counterSensor ++;
183
184 //Helps the user to se how the angle aproaches the target
185 //          if (counter == 100)
186 //          {
187 //            Serial.print(currentAngle);
188 //            Serial.print(" ");
189 //            double plot_time = millis();
190 //            Serial.print(plot_time);
191 //            Serial.print(" ");
192 //            Serial.println(Output);
193 //
194 //            counter = 0;
195 //          //Serial.println(target);
```

```
196 //          }
197 //          counter ++;
198 //*****************************************
199
200           //update the current angle of the shaft to help the
                 user
201           currentAngle = ((float)analogRead(potPin) -
                 POT_VALUE_MIN)/(POT_VALUE_MAX - POT_VALUE_MIN) *
                 MAX_ANGLE;
202
203           // When the error is acceptable for the first time
204           if (abs(target-currentAngle)<PERM_ERROR && time_t ==
                  0)
205           {
206             // The first time the error is acceptable
207             firsttime = millis();
208             // The current time
209             currenttime = millis();
210             time_t = 1;
211           }
212           // If the error still is acceptable change the
                 current time
213           else if(abs(target-currentAngle)<PERM_ERROR &&
                 time_t == 1)
214           {
215            currenttime = millis();
216           }
217           else
218           {
219             // If the error is not acceptable, set time_t=0
220             time_t = 0;
221           }
222
223           // If the error been acceptable in 100 miliseconds
                 the motors will stop
224           if (currenttime-firsttime > 100)
225           {
226
227             Stop();
228             // Return that the sensors didn't find anything
229             return false;
230           }
231
232           // If the arm goes to far this make it to go
```

```
233             backwards again
234           if (currentAngle < target)
235           {
236             SetTurnDirection(right);
237             myPID.SetControllerDirection(DIRECT);
238           }
239           else if (currentAngle > target)
240           {
241             SetTurnDirection(left);
242             myPID.SetControllerDirection(REVERSE);
243           }
244
245         }
246       }
247
248 };
249
250 //*****************************************
251
252
253 // The potentiometers pin definitions
254 int potPinA = 5; // Read the potentiometer value on analog
      pin 5
255
256 int potPinB = 4; // Read the potentiometer value on analog
      pin 4
257
258 int potPinC = 3; // Read the potentiometer value on analog
      pin 3
259
260 // Angular constants for each motor
261 // MIN is the value from the potentiometer where the arm
      reach 0 degrees
262 // MAX is the value from the potentiometer where the arm
      reach 90 degrees
263 double POT_VALUE_MAX_A = 596;
264 double POT_VALUE_MAX_B = 936;
265 double POT_VALUE_MAX_C = 1183;
266
267 double POT_VALUE_MIN_A = 189;
268 double POT_VALUE_MIN_B = 521;
269 double POT_VALUE_MIN_C = 444;
270
```

```
271 // L298 pin defintions
272 int motorA_p1 = 4;
273 int motorA_p2 = 5;
274 int pwmPinA = 8;
275
276 int motorB_p1 = 6;
277 int motorB_p2 = 7;
278 int pwmPinB = 10;
279
280 int motorC_p1 = 11;
281 int motorC_p2 = 12;
282 int pwmPinC = 13;
283
284 // Declare the Servo pin
285 int servoPin = 9;
286
287 // Implementing The pattern calculated from MATLAB
288 // The numbers in the comment below explains the procedure
       of the pattern
289 double angleA1 = 41.0;     // 2
290 double angleA2 = 38.0;     // 5
291 double angleA3 = 33.5;     // 8
292 double angleA4 = 27.6;     // 11
293 double angleA5 = 19.8;     // 14
294 double angleA6 = 55.4;     // 17 Food position
295
296 double angleB1 = -75.3;    // 3
297 double angleB2 = -66.9;    // 6
298 double angleB3 = -58.4;    // 9
299 double angleB4 = -49.3;    // 12
300 double angleB5 = -38.8;    // 15
301 double angleB6 = 30;       // 18 Food position
302
303 double angleC1 = -21.3;    // 1
304 double angleC2 = 21.3;     // 4
305 double angleC3 = -18.0;    // 7
306 double angleC4 = 15.6;     // 10
307 double angleC5 = -13.8;    // 13
308 double angleC6 = 12.3;     // 16
309 double angleC7 = -48.0;    // 19 Food position
310
311 //*****************************************
312
313 // Creates a instance for the servo from the built-in
```

```
    library
Servo Servo1;

// Create instances of the DC motor Class
DCMotor customServoA(motorA_p1, motorA_p2, pwmPinA);
DCMotor customServoB(motorB_p1, motorB_p2, pwmPinB);
DCMotor customServoC(motorC_p1, motorC_p2, pwmPinC);

// The controller parameters dependent the motion
double Kp_A1 = 13, Ki_A1 = 4, Kd_A1 = 4.8;
double Kp_A2 = 6.5, Ki_A2 = 2.5, Kd_A2 = 4.5;
double Kp_A3 = 3, Ki_A3 = 1.3, Kd_A3 = 0.5;

double Kp_B1 = 12, Ki_B1 = 3.6, Kd_B1 = 0.9;
double Kp_B2 = 7, Ki_B2 = 1.2, Kd_B2 = 0.6;
double Kp_B3 = 4.0, Ki_B3 = 1, Kd_B3 = 0.6;

double Kp_C = 0.4, Ki_C = 0.4, Kd_C=0.1;

// sensorStart is always 1 and only used when motor C is
    starting
int sensorStart = 1;
// sensorSearch is initiatet as false
boolean sensorSearch = false;

void setup()
{
  Serial.begin(115200);
  // Attach the servo to the used pin number
  Servo1.attach(servoPin);
}

//*****************************************

void loop()
{
  // Converting the analog value from the potentiometer to
     degrees and printing it in the monitor
  Serial.println("Potentiometer A");
  double AngleA = ((float)analogRead(potPinA) -
     POT_VALUE_MIN_A)/(POT_VALUE_MAX_A - POT_VALUE_MIN_A) *
     MAX_ANGLE;
  Serial.println(AngleA);
  Serial.println(analogRead(potPinA));
```

```
353
354     Serial.println("Potentiometer B");
355     double AngleB = ((float)analogRead(potPinB) -
            POT_VALUE_MIN_B)/(POT_VALUE_MAX_B - POT_VALUE_MIN_B) *
            MAX_ANGLE;
356     Serial.println(AngleB);
357     Serial.println(analogRead(potPinB));
358     Servo1.write(AngleB*-1+5);
359
360     Serial.println("Potentiometer C");
361     double AngleC = ((float)analogRead(potPinC) -
            POT_VALUE_MIN_C)/(POT_VALUE_MAX_C - POT_VALUE_MIN_C) *
            MAX_ANGLE;
362     Serial.println(AngleC);
363     Serial.println(analogRead(potPinC));
364
365     Serial.println("Type any button to start");
366     delay(1000);
367
368     // If typing anything in the monitor the pattern will
            start
369     if (Serial.available())
370     {
371       // test is used to be able to pause the program
372       int test = Serial.read();
373       while(true)
374       {
375         Serial.println("C1");
376         // Parameters is sent to GoToAngle
377         customServoC.GoToAngle(angleC1, potPinC,
                POT_VALUE_MAX_C, POT_VALUE_MIN_C, Kp_C, Ki_C, Kd_C)
                ;
378
379         Serial.println("A1");
380         customServoA.GoToAngle(angleA1, potPinA,
                POT_VALUE_MAX_A, POT_VALUE_MIN_A, Kp_A3, Ki_A3,
                Kd_A3);
381         Serial.println("B1");
382         customServoB.GoToAngle(angleB1, potPinB,
                POT_VALUE_MAX_B, POT_VALUE_MIN_B, Kp_B3, Ki_B3,
                Kd_B3);
383         // The servo turns to the correct angle and is only
                dependent on motor B
384         Servo1.write(angleB1*-1-5);
```

```
385         delay(800);
386
387         Serial.println("C2");
388         sensorSearch = customServoC.GoToAngle(angleC2, potPinC
                , POT_VALUE_MAX_C, POT_VALUE_MIN_C, Kp_C, Ki_C,
                Kd_C, sensorStart); // sensorStart is sent in to
                start the sensor
389         if (sensorSearch == true) // If the sensor detected
                something, sensorSearch is true and the while loop
                will break
390         {
391           Serial.println("Food found");
392           delay(5000);
393           break;
394         }
395
396         Serial.println("A2");
397         customServoA.GoToAngle(angleA2, potPinA,
                POT_VALUE_MAX_A, POT_VALUE_MIN_A, Kp_A1, Ki_A1,
                Kd_A1);
398         Serial.println("B2");
399         customServoB.GoToAngle(angleB2, potPinB,
                POT_VALUE_MAX_B, POT_VALUE_MIN_B, Kp_B1, Ki_B1,
                Kd_B1);
400         Servo1.write(angleB2*-1-5);
401         delay(800);
402
403         Serial.println("C3");
404         sensorSearch = customServoC.GoToAngle(angleC3, potPinC
                , POT_VALUE_MAX_C, POT_VALUE_MIN_C, Kp_C, Ki_C,
                Kd_C, sensorStart);
405         if (sensorSearch == true)
406         {
407           Serial.println("Food found");
408           delay(5000);
409           break;
410         }
411
412         Serial.println("A3");
413         customServoA.GoToAngle(angleA3, potPinA,
                POT_VALUE_MAX_A, POT_VALUE_MIN_A, Kp_A1, Ki_A1,
                Kd_A1);
414         Serial.println("B3");
415         customServoB.GoToAngle(angleB3, potPinB,
```

```
        POT_VALUE_MAX_B, POT_VALUE_MIN_B, Kp_B1, Ki_B1,
        Kd_B1);
416     Servo1.write(angleB3*-1-5);
417     delay(800);
418
419     Serial.println("C4");
420     sensorSearch = customServoC.GoToAngle(angleC4, potPinC
        , POT_VALUE_MAX_C, POT_VALUE_MIN_C, Kp_C, Ki_C,
        Kd_C, sensorStart);
421     if (sensorSearch == true)
422     {
423       Serial.println("Food found");
424       delay(5000);
425       break;
426     }
427
428     Serial.println("A4");
429     customServoA.GoToAngle(angleA4, potPinA,
        POT_VALUE_MAX_A, POT_VALUE_MIN_A, Kp_A2, Ki_A2,
        Kd_A2);
430     Serial.println("B4");
431     customServoB.GoToAngle(angleB4, potPinB,
        POT_VALUE_MAX_B, POT_VALUE_MIN_B, Kp_B1, Ki_B1,
        Kd_B1);
432     Servo1.write(angleB4*-1-5);
433     delay(800);
434
435     Serial.println("C5");
436     sensorSearch = customServoC.GoToAngle(angleC5, potPinC
        , POT_VALUE_MAX_C, POT_VALUE_MIN_C, Kp_C, Ki_C,
        Kd_C, sensorStart);
437     if (sensorSearch == true)
438     {
439       Serial.println("Food found");
440       delay(5000);
441       break;
442     }
443
444     Serial.println("A5");
445     customServoA.GoToAngle(angleA5, potPinA,
        POT_VALUE_MAX_A, POT_VALUE_MIN_A, Kp_A2, Ki_A2,
        Kd_A2);
446     Serial.println("B5");
447     customServoB.GoToAngle(angleB5, potPinB,
```

```
448             POT_VALUE_MAX_B, POT_VALUE_MIN_B, Kp_B1, Ki_B1,
                    Kd_B1);
449         Servo1.write(angleB5*-1-5);
450         delay(800);
451
452         Serial.println("C6");
453         sensorSearch = customServoC.GoToAngle(angleC6, potPinC
                , POT_VALUE_MAX_C, POT_VALUE_MIN_C, Kp_C, Ki_C,
                    Kd_C, sensorStart);
454         if (sensorSearch == true)
455         {
456           Serial.println("Food found");
457           delay(5000);
458           break;
459         }
460
461         Serial.println("The robot did not find food..");
462         // If the robot didn't find anything it pauses until
                something is typed in the monitor
463         while (Serial.available() == 0) //Pause the program
                until something is typed in the monitor
464           {}
465         test = Serial.read();
466       }
467
468       Serial.println("B6");
469       customServoB.GoToAngle(angleB6, potPinB, POT_VALUE_MAX_B
              , POT_VALUE_MIN_B, Kp_B2, Ki_B2, Kd_B2);
470       Serial.println("A6");
471       customServoA.GoToAngle(angleA6, potPinA, POT_VALUE_MAX_A
              , POT_VALUE_MIN_A, Kp_A3, Ki_A3, Kd_A3);
472
473       Serial.println("C7");
474       customServoC.GoToAngle(angleC7, potPinC, POT_VALUE_MAX_C
              , POT_VALUE_MIN_C, Kp_C, Ki_C, Kd_C);
475
476       Serial.println("Press any button for more imaginary food
              .");
477       while (Serial.available() == 0)
478           {}
479     }
480 }
481
```

```
482 //**********************************The code below is used
    for tuning the degree positions of the robot arms
    ************************
483
484 //  if (Serial.available())
485 //  {
486 //    int chooseMotor = Serial.read();
487 //    if (chooseMotor == 'A')
488 //    {
489 //      while (Serial.available() == 0)
490 //      {}
491 //      float wantedAngleA = Serial.parseFloat();
492 //      Serial.println(wantedAngleA);
493 //      customServoA.GoToAngle(wantedAngleA, potPinA,
    POT_VALUE_MAX_A, POT_VALUE_MIN_A, Kp_A3, Ki_A3, Kd_A3);
494 //    }
495 //
496 //    if (chooseMotor == 'B')
497 //    {
498 //      while (Serial.available() == 0)
499 //      {}
500 //      float wantedAngleB = Serial.parseFloat();
501 //      Serial.println(wantedAngleB);
502 //      customServoB.GoToAngle(wantedAngleB, potPinB,
    POT_VALUE_MAX_B, POT_VALUE_MIN_B, Kp_B3, Ki_B3, Kd_B3);
503 //      Servo1.write(wantedAngleB*-1);
504 //    }
505 //    if (chooseMotor == 'C')
506 //    {
507 //      while (Serial.available() == 0)
508 //      {}
509 //      float wantedAngleC = Serial.parseFloat();
510 //      Serial.println(wantedAngleC);
511 //      customServoC.GoToAngle(wantedAngleC, potPinC,
    POT_VALUE_MAX_C, POT_VALUE_MIN_C, Kp_C, Ki_C, Kd_C,
    sensorStart);
512 //    }
513 //    if(chooseMotor == 'D')
514 //    {
515 //      Serial.print("Servo");
516 //      while (Serial.available() == 0)
517 //      {}
518
519 //      int wantedAngleD = Serial.parseInt();
```

```
520 //        Servo1.write(wantedAngleD);
521 //
522 //      }
523 //      if(chooseMotor == 'E')
524 //      {
525 //        const int trigPin = 3;
526 //        const int echoPin = 2;
527 //        // defines variables
528 //        long duration;
529 //        pinMode(trigPin, OUTPUT); // Sets the trigPin as an
    Output
530 //        pinMode(echoPin, INPUT); // Sets the echoPin as an
    Input
531 //
532 //        while(1==1)
533 //        {
534 //          // Clears the trigPin
535 //          digitalWrite(trigPin, LOW);
536 //          delayMicroseconds(2);
537 //          // Sets the trigPin on HIGH state for 10 micro
    seconds
538 //          digitalWrite(trigPin, HIGH);
539 //          delayMicroseconds(10);
540 //          digitalWrite(trigPin, LOW);
541 //          // Reads the echoPin, returns the sound wave
    travel time in microseconds
542 //          duration = pulseIn(echoPin, HIGH);
543 //          // Calculating the distance
544 //          // Prints the distance on the Serial Monitor
545 //          Serial.print("Distance: ");
546 //          Serial.println(duration);
547 //          delay(50);
548 //        }
549 //      }
550 //   }
551 //}
552
553 //*****************************************
```

# Appendix C

# Matlab Code

## Matlab program for inverse and forward kinematics

```
%Feeding Robot
%Filip Malberg & Linda Truong
%TRITA MMK 2017:26 MDAB 644
%KTH, ITM, MDA

clear all;
close all;
clc;
%% Inverse kinematics calculating the angles
% Desired coordinates (mm)
Px = 215;                   % x-coordinate
Py = -70;                   % y-coordinate
Pz = 100;                   % z-coordinate

theta = atand(Py/Px);
L1 = 195;
L2 = 185;
hz = 150;
L3 = sqrt(Px^2 + Py^2 + (Pz-hz)^2);                 % L3 is the resultant of L1 and L2
Beta = acosd((L2^2 + L1^2 - L3^2)/(2*L1*L2));       % Beta, a1, a2 is angles
a1 = atand((Pz-hz)/sqrt(Px^2+Py^2));
a2 = acosd((L1^2 + L3^2 - L2^2)/(2*L1*L3));
phi1 = atand((Pz-hz)/sqrt(Px^2+Py^2)) + acosd((L1^2 + L3^2 - L2^2)/(2*L1*L3));
phi2 = acosd((L2^2 + L1^2 - L3^2)/(2*L1*L2)) + atand((Pz-hz)/sqrt(Px^2+Py^2)) + acosd((L1^2 + L3^2 - L2^2)/(2*L1*L3)) - 180;

disp(phi1);
disp(phi2);
disp(theta);

%% Forward kinematics calculating the coordinates
% Desired angles (degrees)
AngleA = 34;
AngleB = -79;
AngleC = -30;

X = cosd(AngleC)*(L1*cosd(AngleA)+L2*cosd(AngleB));
Y = sind(AngleC)*(L1*cosd(AngleA)+L2*cosd(AngleB));
```

```
38 Z = L1*sind(AngleA) + L2*sind(AngleB) + hz;
39
40 disp(X);
41 disp(Y);
42 disp(Z);
```

## Matlab program for torque constant

```
%Feeding Robot
%Filip Malberg & Linda Truong
%TRITA MMK 2017:26 MDAB 644
%KTH, ITM, MDA

%Motor A, motor B
%Measured time [s]
t_test_1 = [4.69 4.69 4.87 4.87 4.87];
%Mean value of the time [s]
t1 = sum(t_test_1)/5;
%Distance of the wire [m]
s_1 = 38 * 10^-2;
%Radius of the 3D printed part attached to the motor axle [m
    ]
r_1 = (21.5/2) * 10^-3;
%Velocity [m/s]
v_1 = s_1/t1;
%Angular velocity (before gearing) [rad(s^-1)]
omega_1 = v_1/r_1;
%Voltage [V]
U_1 = 12;
%Current [A]
I_1 = 1.33;
%Resistance [Ohm]
R_min_1 = 2.3;
%Angular velocity (after gearing) [rad(s^1)]
omega_rotor_1 = omega_1*60;
%Torque constant [V/rad(s^1)]
K_large_minR = (U_1 - R_min_1*I_1)/omega_rotor_1


%Motor C
%Measured time [s]
t_test_2 = [4.21 3.95 4 3.85 3.90];
%Mean value of the time [s]
t_2 = sum(t_test_2)/5;
%Distance of the wire [m]
s_2 =  38 * 10^-2;
%Radius of the 3D printed part attached to the motor axle [m
    ]
r_2 =  (22.2/2) * 10^-3;
%Velocity [m/s]
```

```
41 v_2 = s_2/t_2;
42 %Angular velocity (before gearing) [rad(s^-1)]
43 omega_2 = v_2/r_2;
44 %Voltage [V]
45 U_2 = 12;
46 %Current [A]
47 I_2 = 0.108;
48 %Resistance [Ohm]
49 R_min_2 = 53.6;
50 %Angular velocity (after gearing) [rad(s^1)]
51 omega_rotor_2 = omega_2*78.9;
52 %Torque constant [V/rad(s^1)]
53 K_small_minR = (U_2 - R_min_2*I_2)/omega_rotor_2
```

## Matlab program for calculations of PID-parameters for motor A

```
%Feeding Robot
%Filip Malberg & Linda Truong
%TRITA MMK 2017:26 MDAB 644
%KTH, ITM, MDA

%Constants for motor A
K_large = 0.0202;                                   %[Nm/A]
J_rotor_largeA = 483.693*10^-3*(10^-2)^2;   %[kgm^2]
R_largeA = 2.3;                                     %[A]
n_m_largeA = 60;
n_system_largeA = 1;
J_system_largeA = 1216.913*10^-3*(10^-2)^2;%[kgm^2]


N_largeA = (n_m_largeA*n_system_largeA)^2;
jA_largeA = J_rotor_largeA*R_largeA;
jB_largeA = J_system_largeA*R_largeA;

%Parameters for motor A
I = (49.2372*(jB_small+jA_small))/(K_large*N_largeA)
P = (40.3011*(jB_small+jA_small))/(K_large*N_largeA)
D = (10.9956*(jB_small+jA_small) - (K_large^2))/(K_large*
    N_largeA)

L = 0;   %Neglected inductance due to small value
b = 0;   %Neglected viscous friction

% G_p = K / (As^3 + Bs^2 + Cs)
A = J_rotor_largeA*L + (J_system_largeA*L)/((n_m_largeA*
    n_system_largeA)^2);
B = J_rotor_largeA*R_largeA + (J_system_largeA*R_largeA)/((
    n_m_largeA*n_system_largeA)^2) + b*L;
C = K_large^2 + b*R_largeA;

p_motorA = [A B C 0];
poles_motorA = roots(p_motorA)

% fastpole = -3.6652
% (s+4.6683)^3 = s^3 + 10.9956s^2 + 40.3011s + 49.2372
```

TRITA MMK 2017:26 MDAB 644